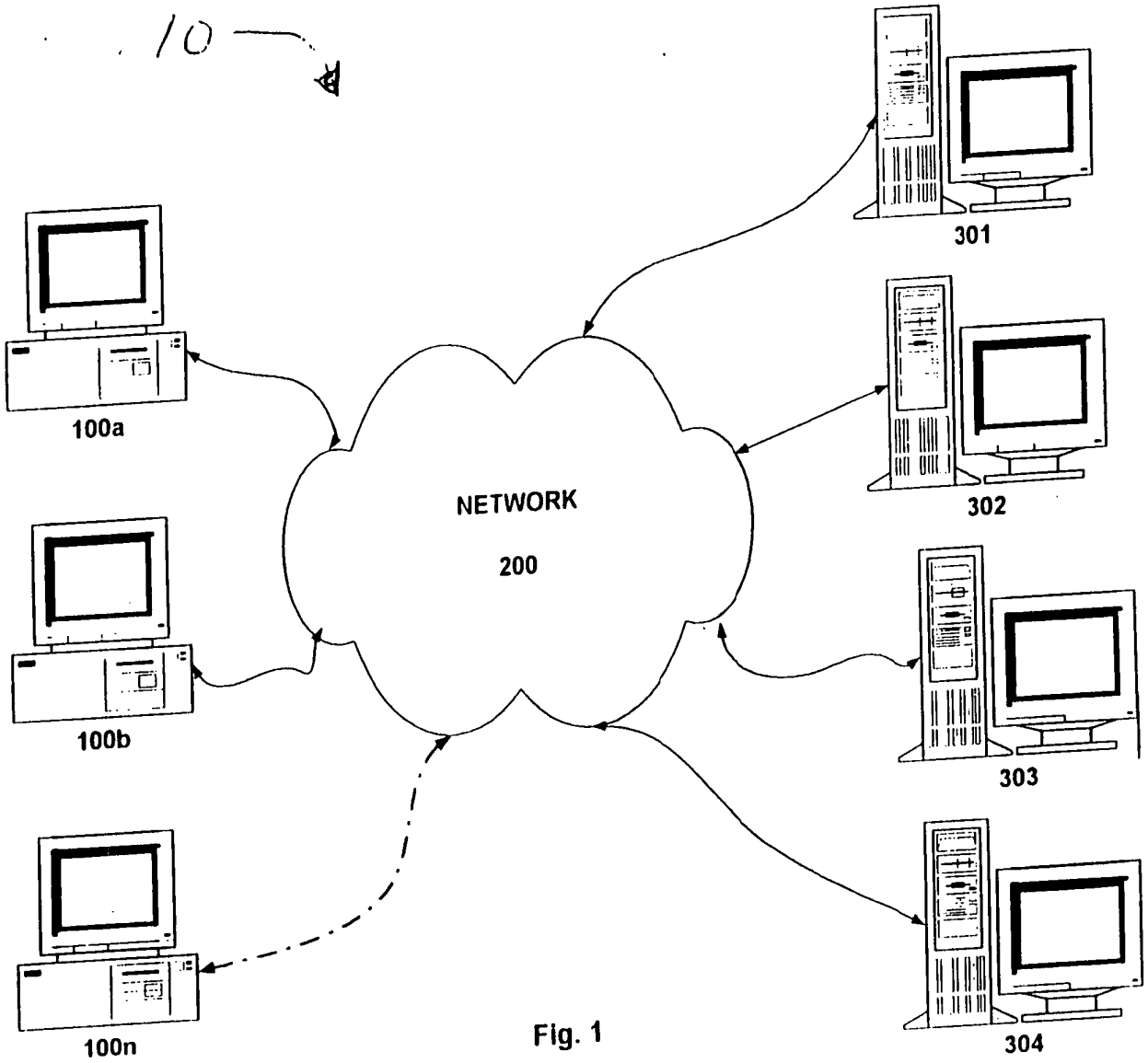

Fig. 1

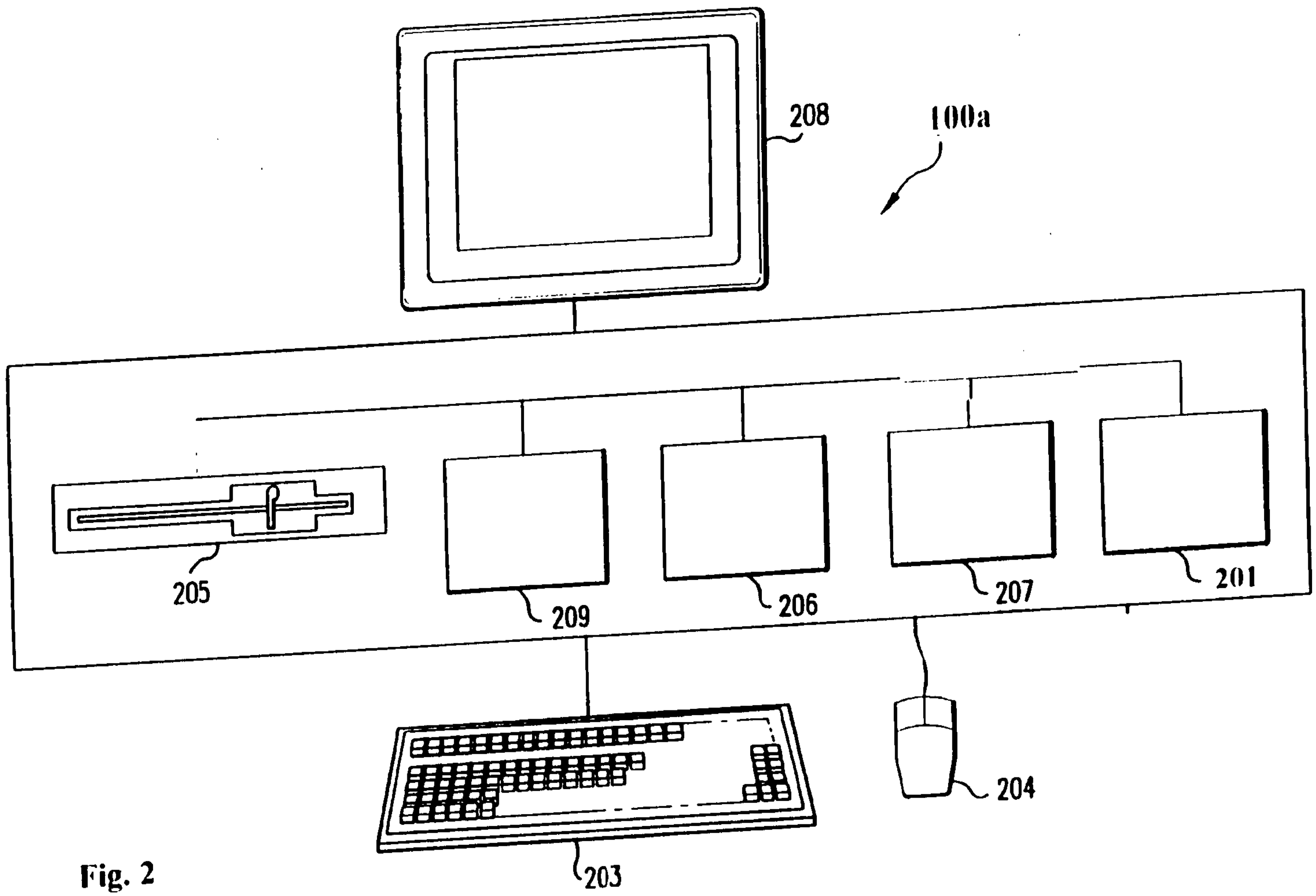

Fig. 2

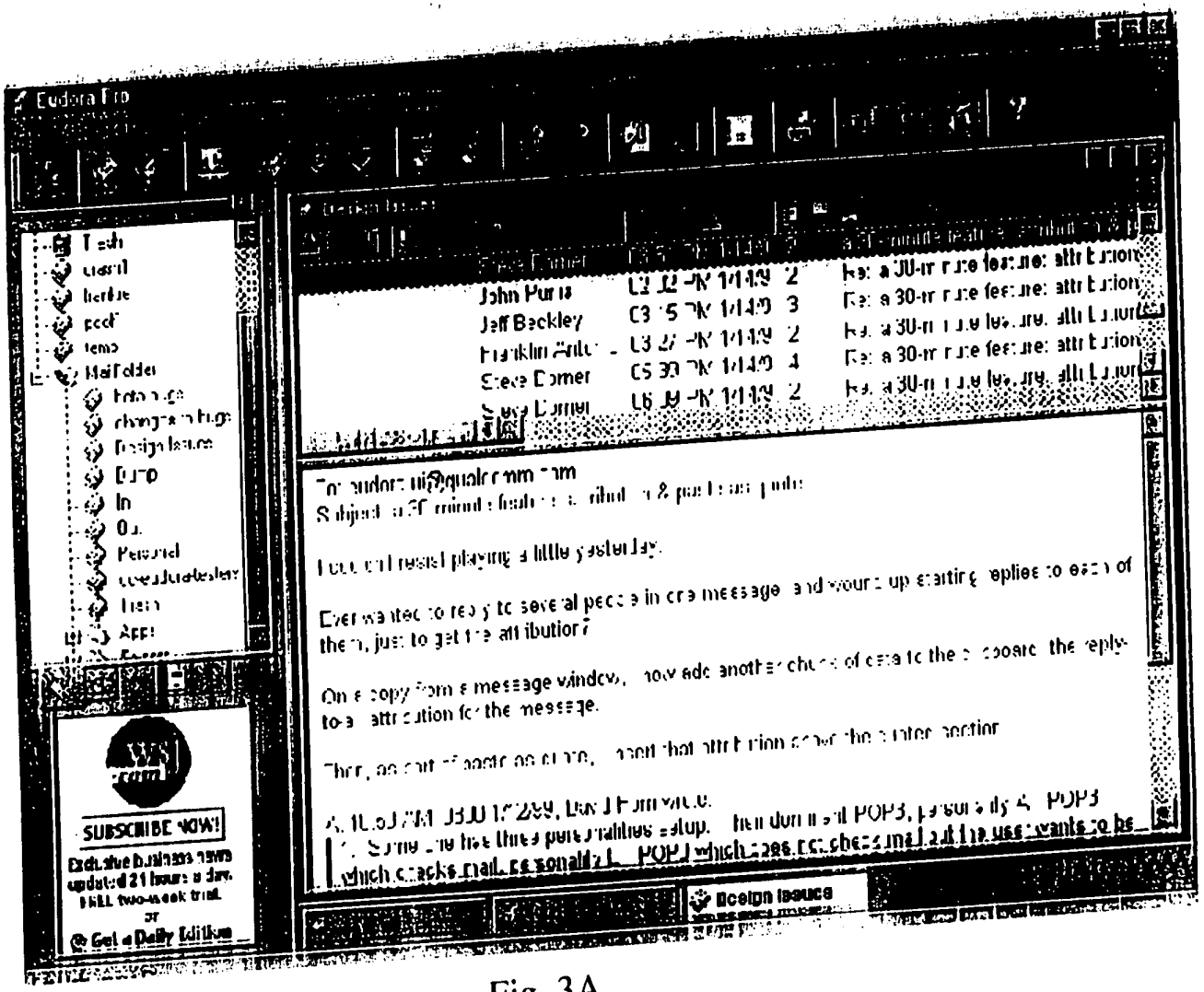
Eudora Pro
Trash
Mail Folder
In
Out
Personal
SUBSCRIBE NOW!
Design Issues

Fig. 3A

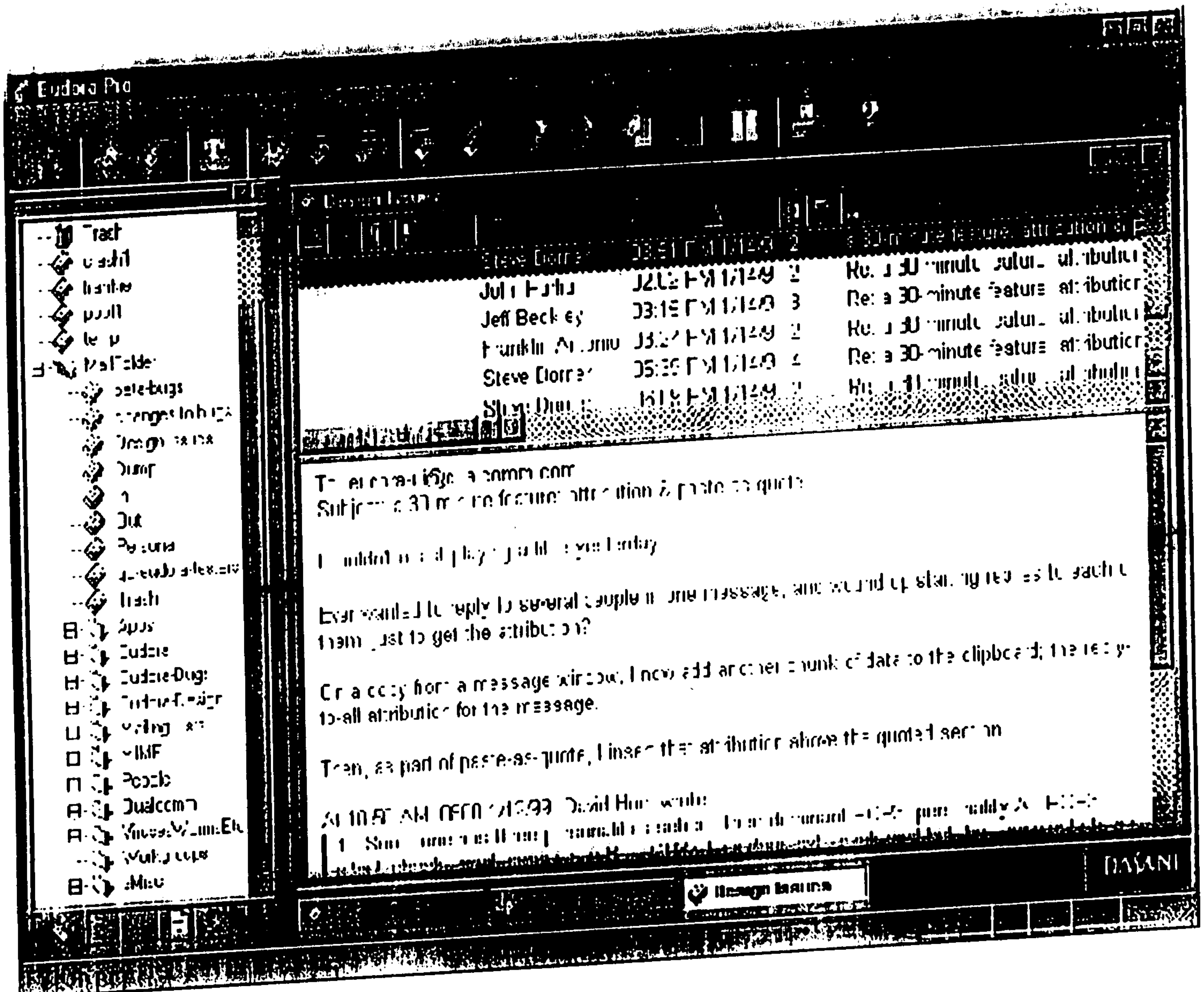
Eudora Pro
Trash
Jeff Beckley
Steve Dorner
Re: a 30-minute feature attribution

Fig. 3B

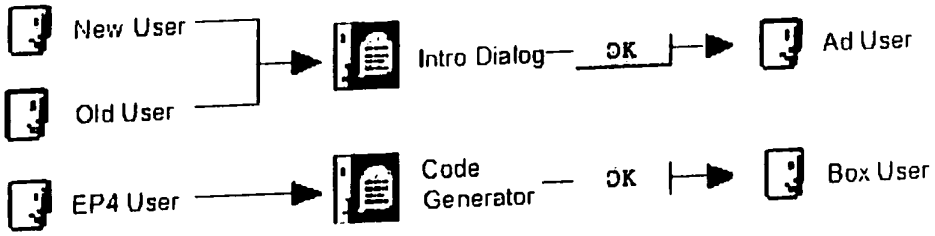

Fig. 4A

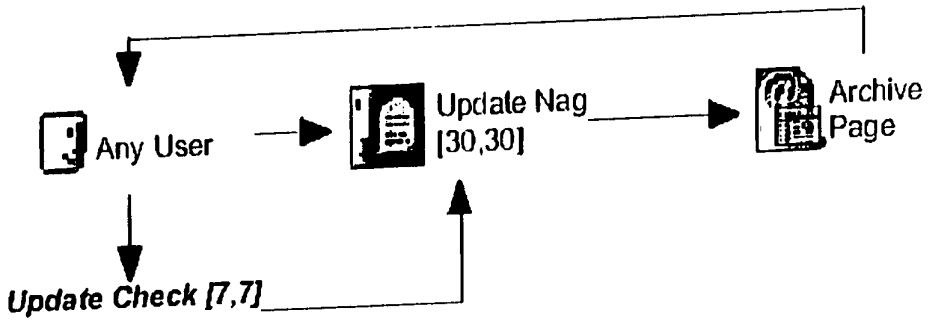

Fig. 7A

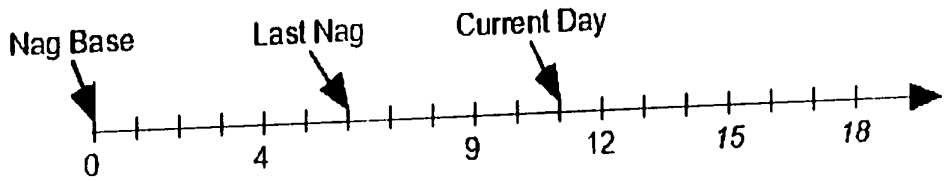

Fig. 11

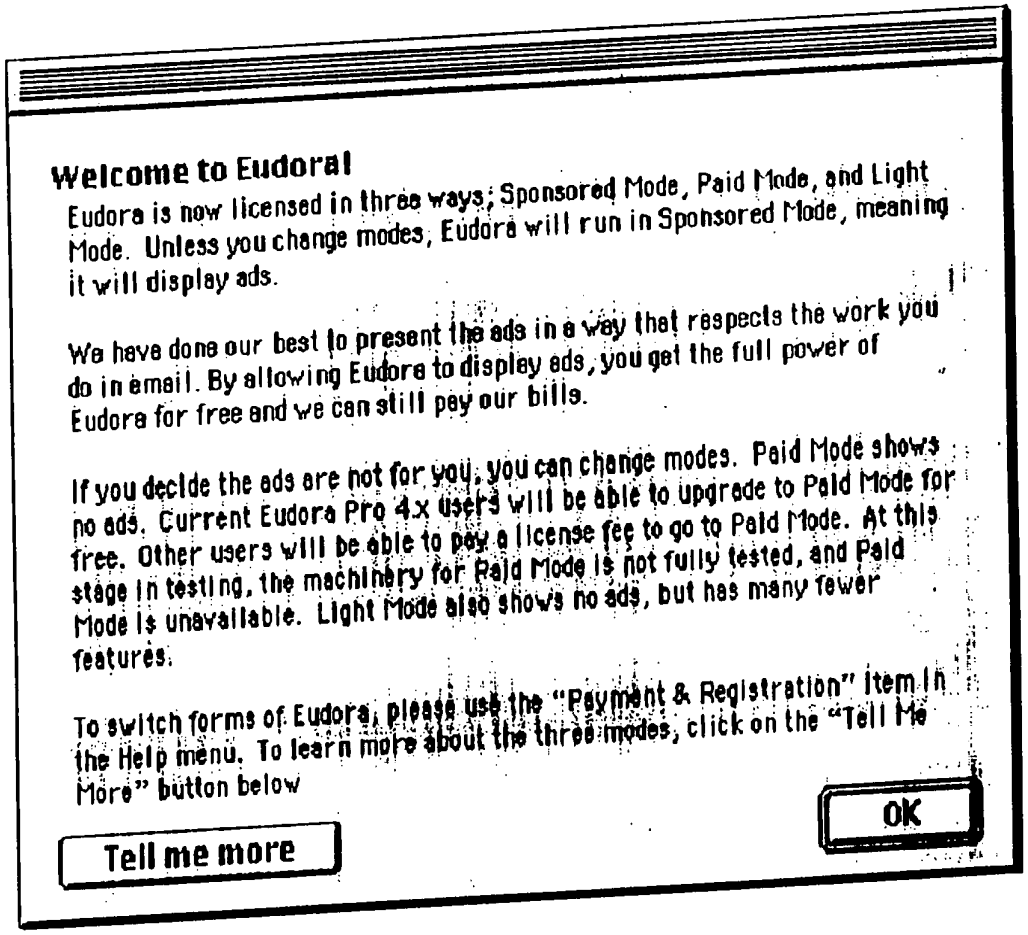
Welcome to Eudora!
Eudora is now licensed in three ways; Sponsored Mode, Paid Mode, and Light Mode. Unless you change modes, Eudora will run in Sponsored Mode, meaning it will display ads.
We have done our best to present the ads in a way that respects the work you do in email. By allowing Eudora to display ads, you get the full power of Eudora for free and we can still pay our bills.
If you decide the ads are not for you, you can change modes. Paid Mode shows no ads. Current Eudora Pro 4.x users will be able to upgrade to Paid Mode for free. Other users will be able to pay a license fee to go to Paid Mode. At this stage in testing, the machinery for Paid Mode is not fully tested, and Paid Mode is unavailable. Light Mode also shows no ads, but has many fewer features.
To switch forms of Eudora, please use the "Payment & Registration" item in the Help menu. To learn more about the three modes, click on the "Tell Me More" button below
Tell me more
OK

Fig. 4B

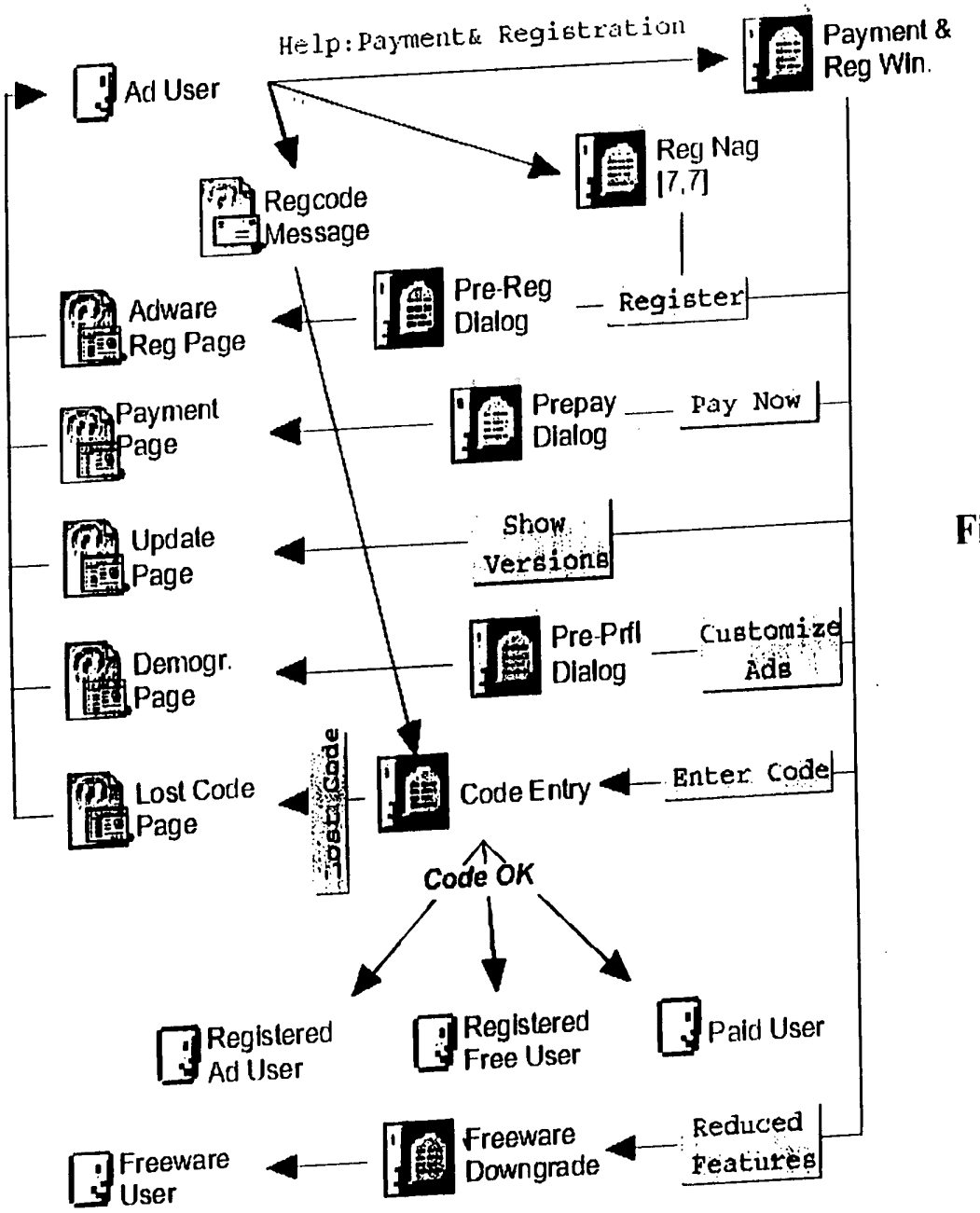

Fig. 5A

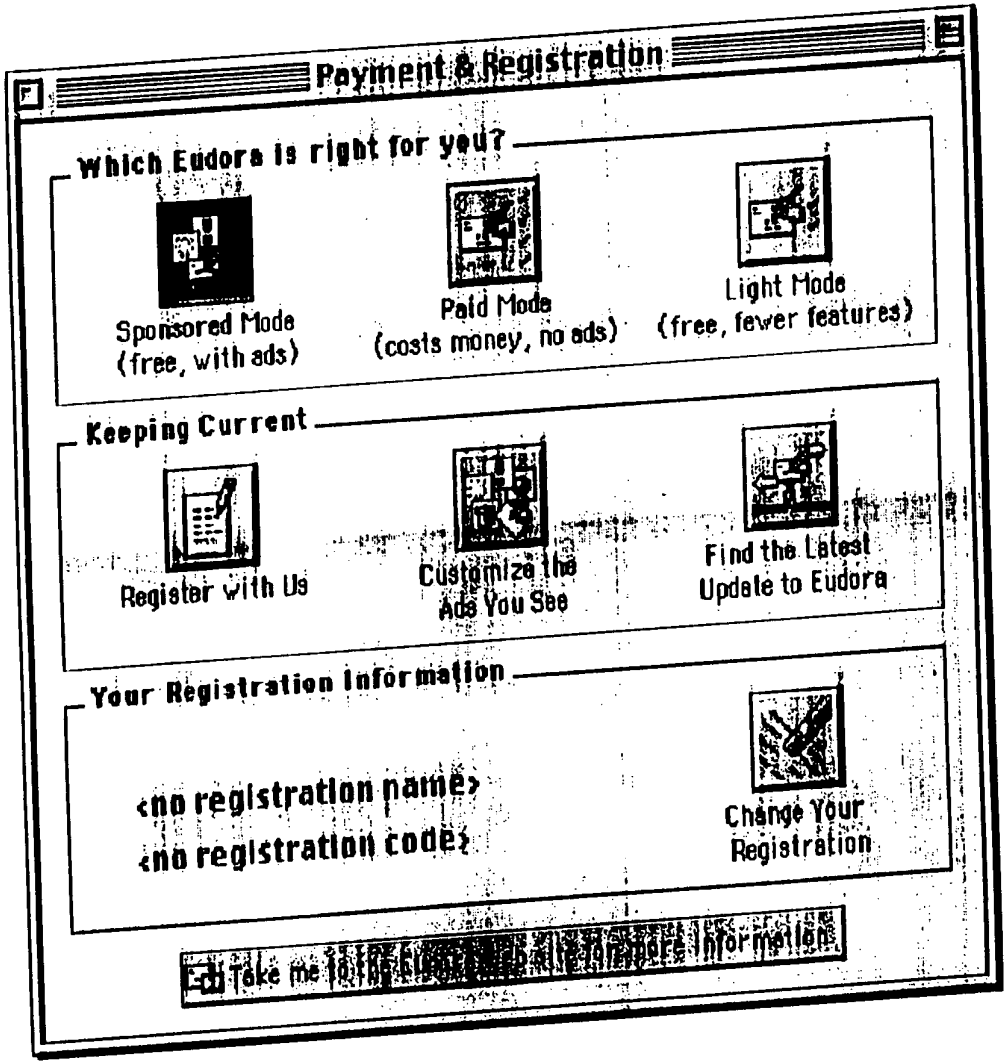

Fig. 5B

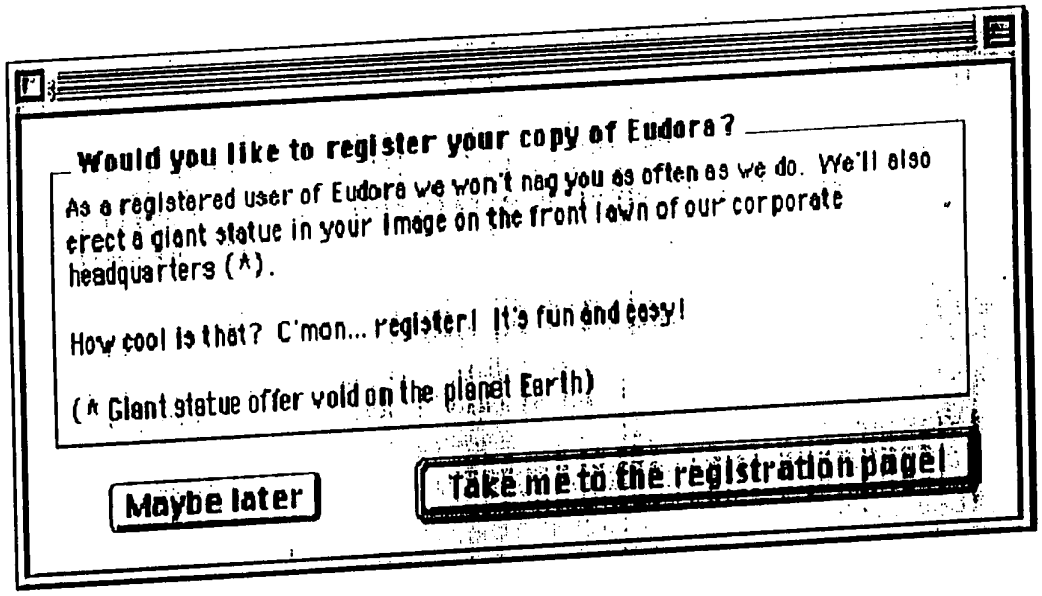

Fig. 5C

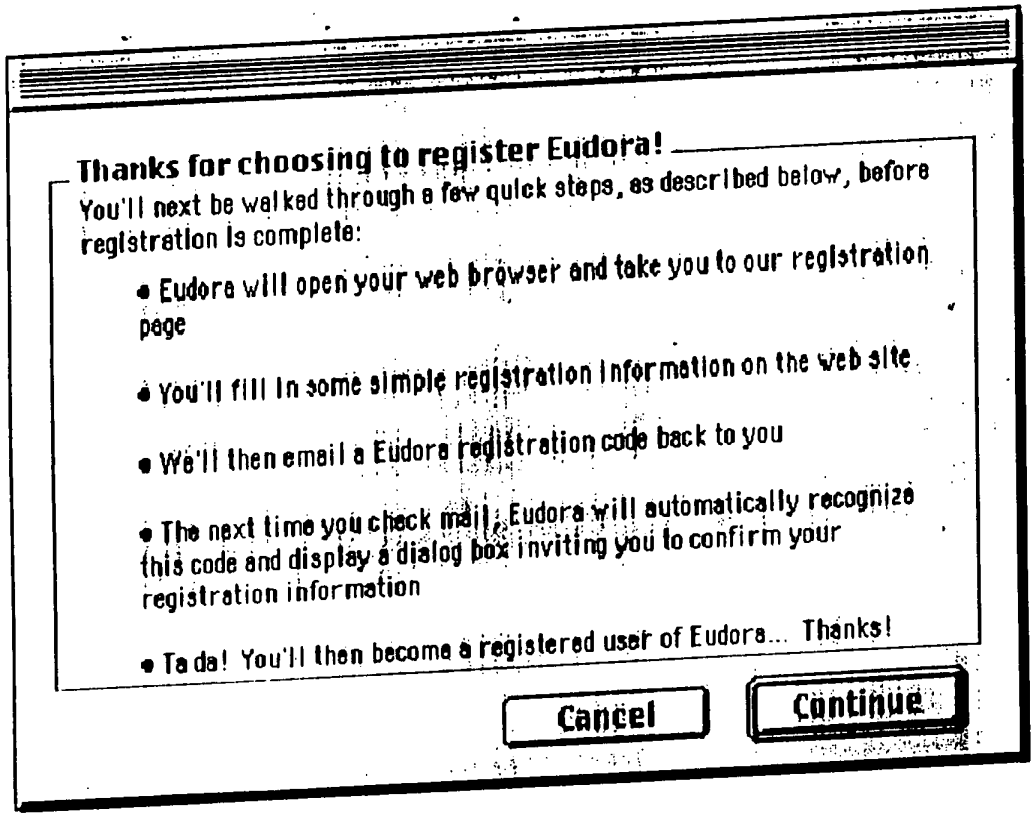

Fig. 5D

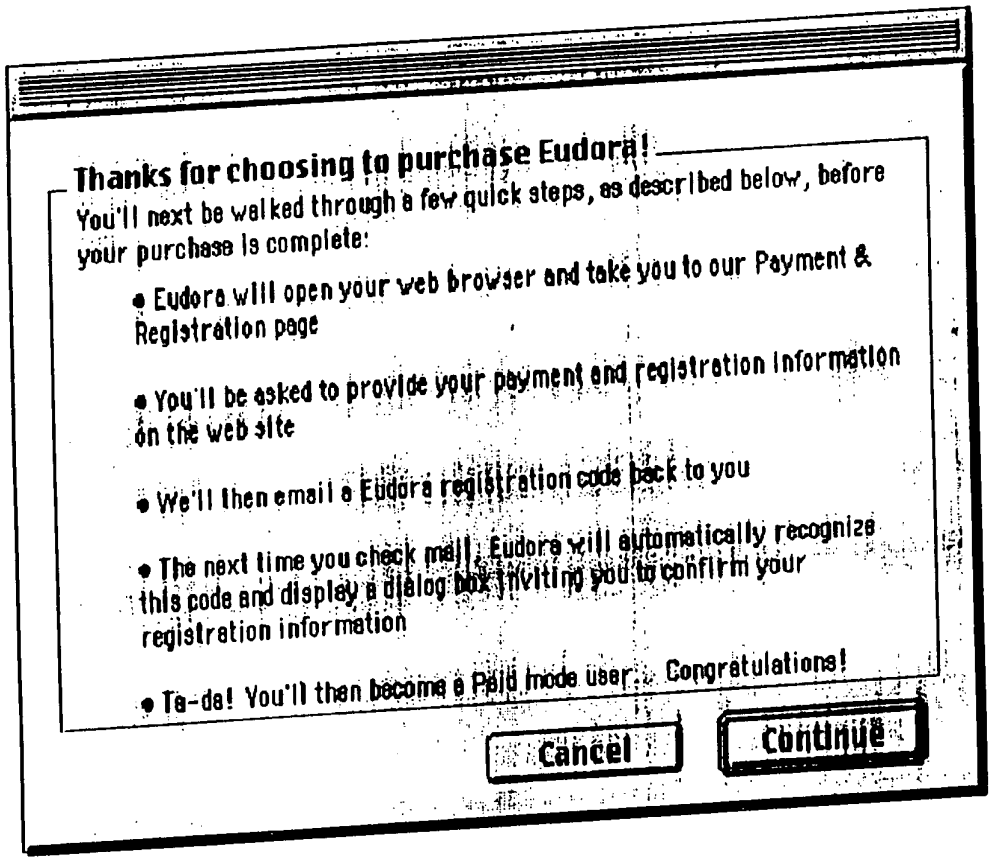
Thanks for choosing to purchase Eudora!
You'll next be walked through a few quick steps, as described below, before your purchase is complete:
• Eudora will open your web browser and take you to our Payment & Registration page
• You'll be asked to provide your payment and registration information on the web site
• We'll then email a Eudora registration code back to you
• The next time you check mail, Eudora will automatically recognize this code and display a dialog box inviting you to confirm your registration information
• Ta-da! You'll then become a Paid mode user. Congratulations!
Cancel
Continue

Fig. 5E

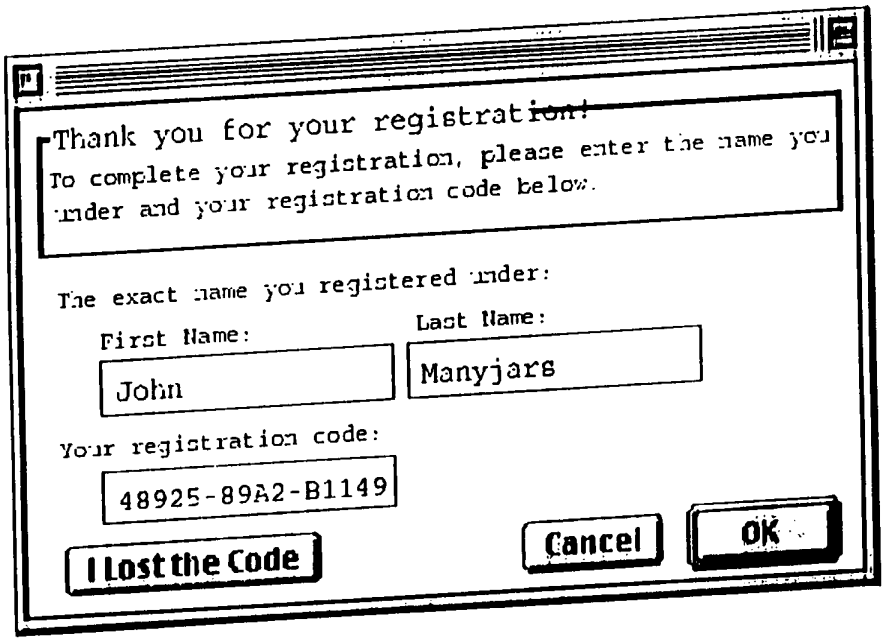

Fig. 5F

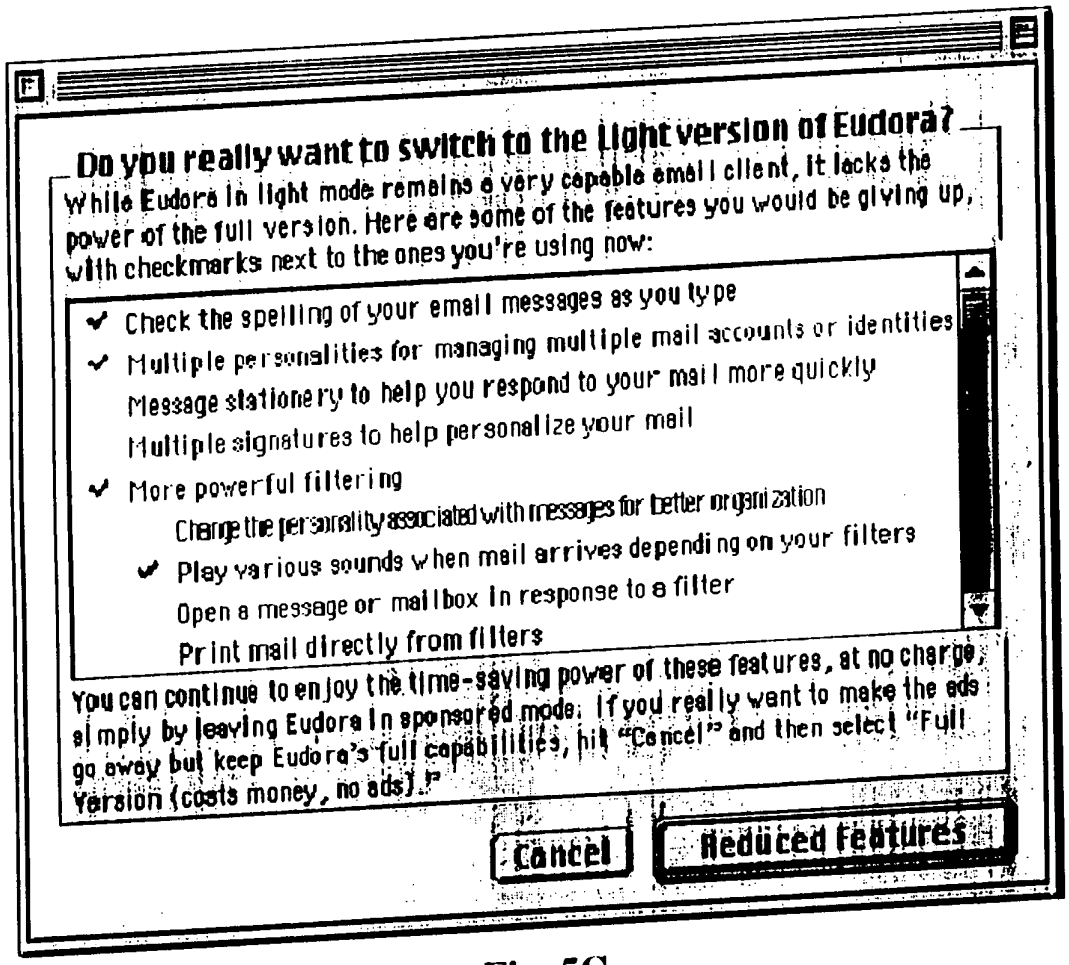

Fig. 5G

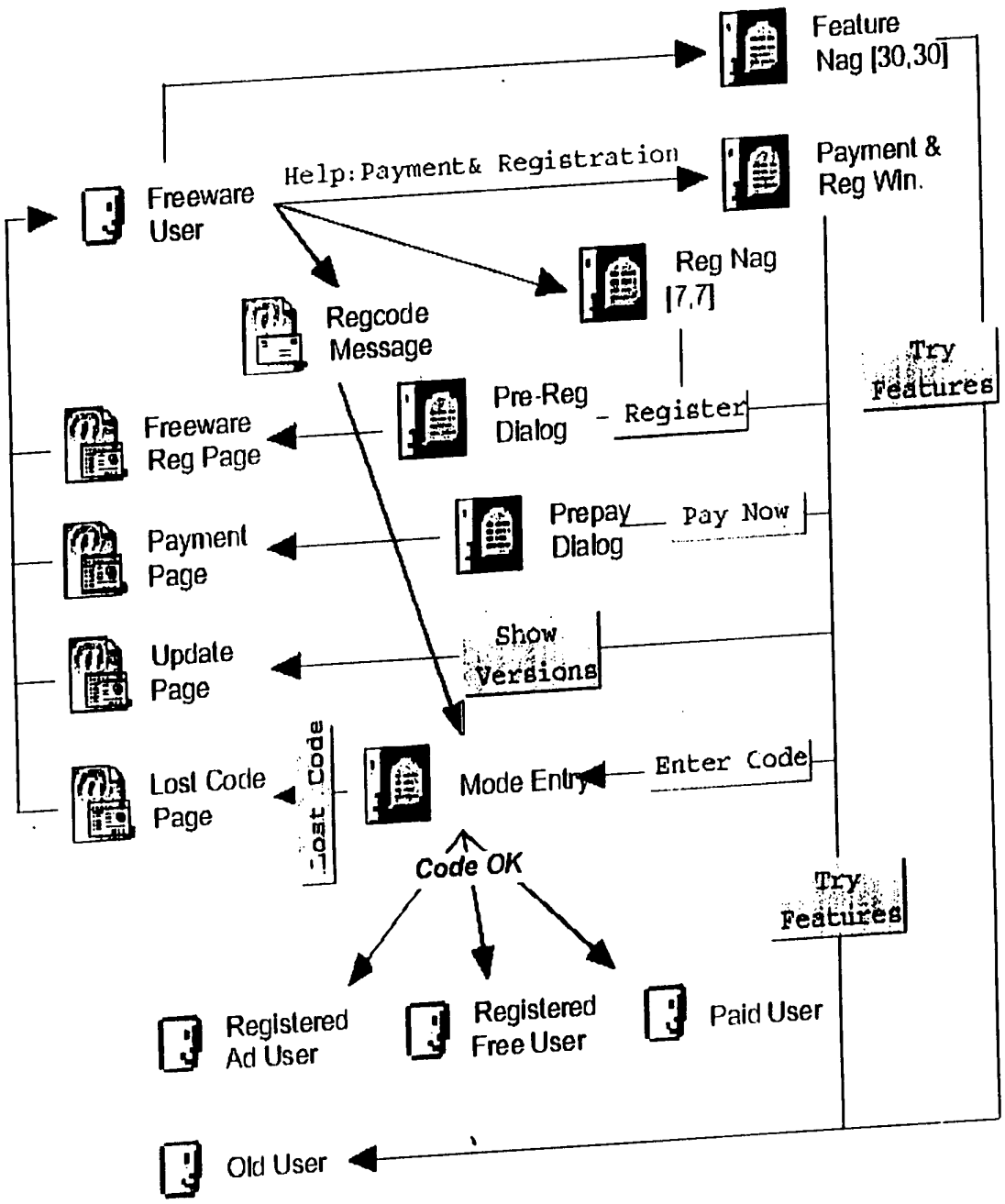

Fig. 6A

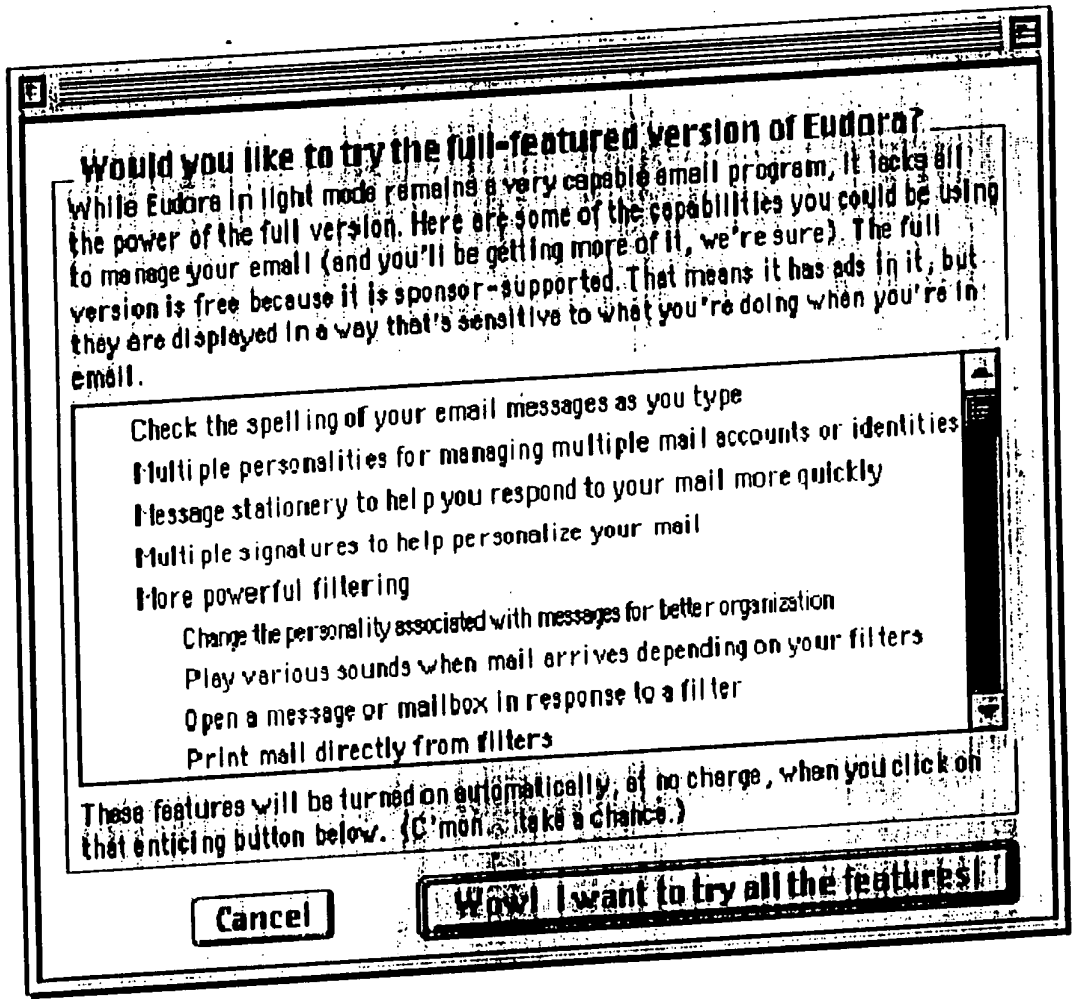

Fig. 6B

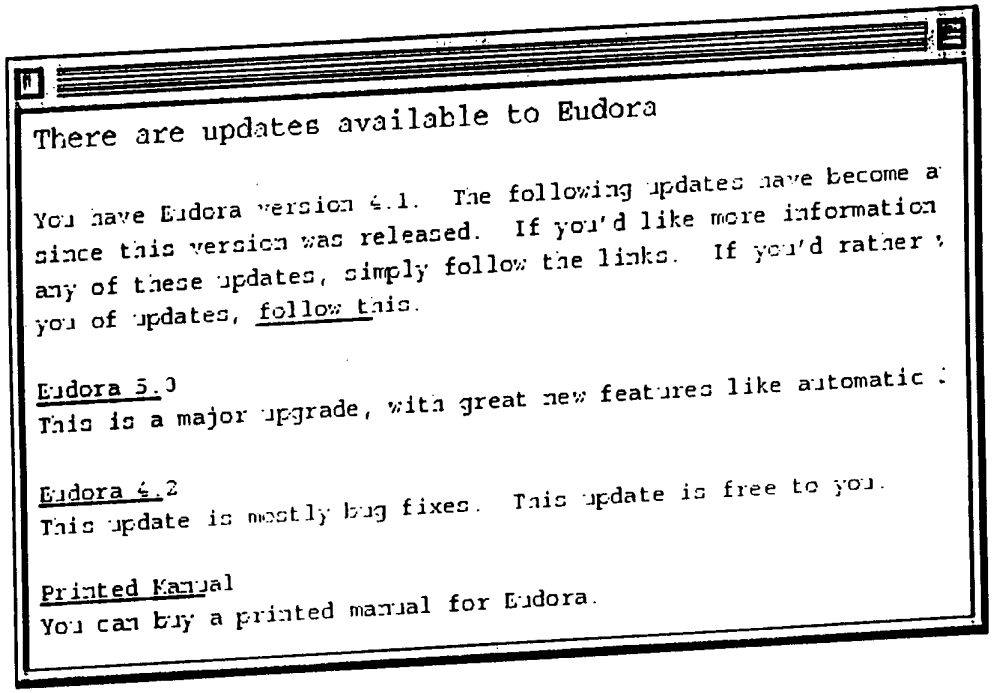
There are updates available to Eudora
You have Eudora version 4.1. The following updates have become a
since this version was released. If you'd like more information
any of these updates, simply follow the links. If you'd rather
you of updates, follow this.
Eudora 5.0
This is a major upgrade, with great new features like automatic
Eudora 4.2
This update is mostly bug fixes. This update is free to you.
Printed Manual
You can buy a printed manual for Eudora.

Fig. 7B

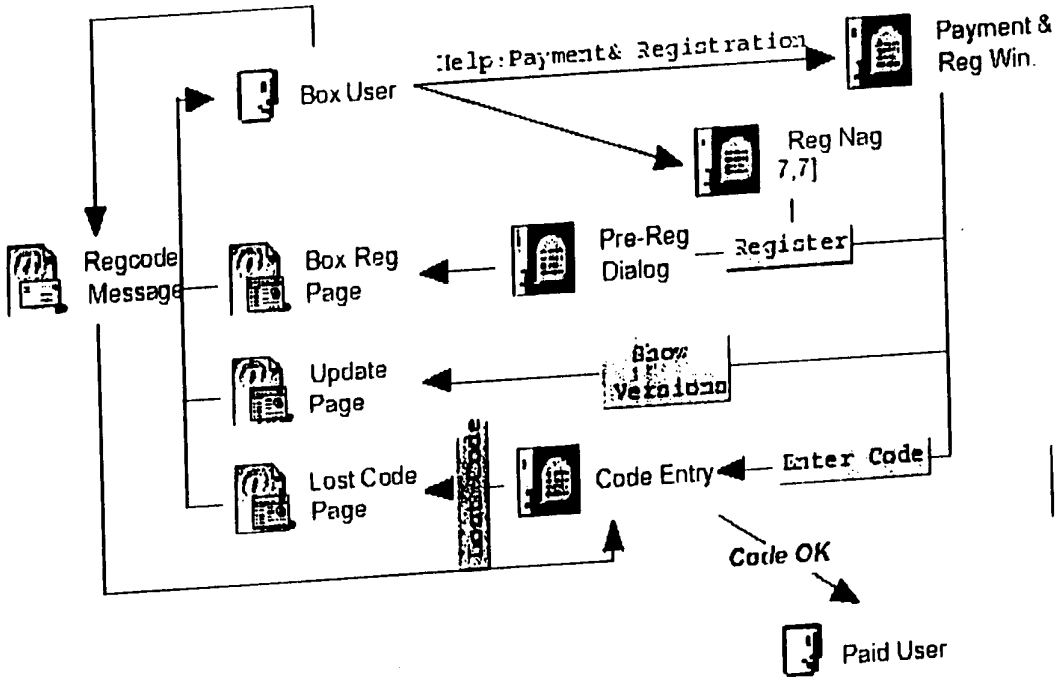

Fig. 8

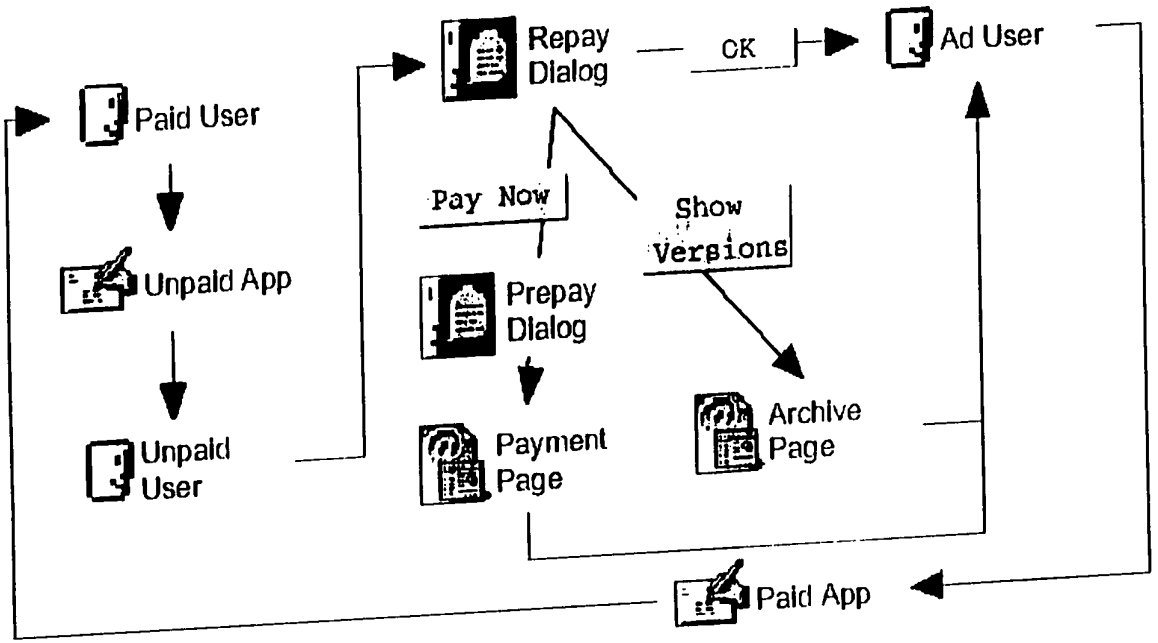

Fig. 9

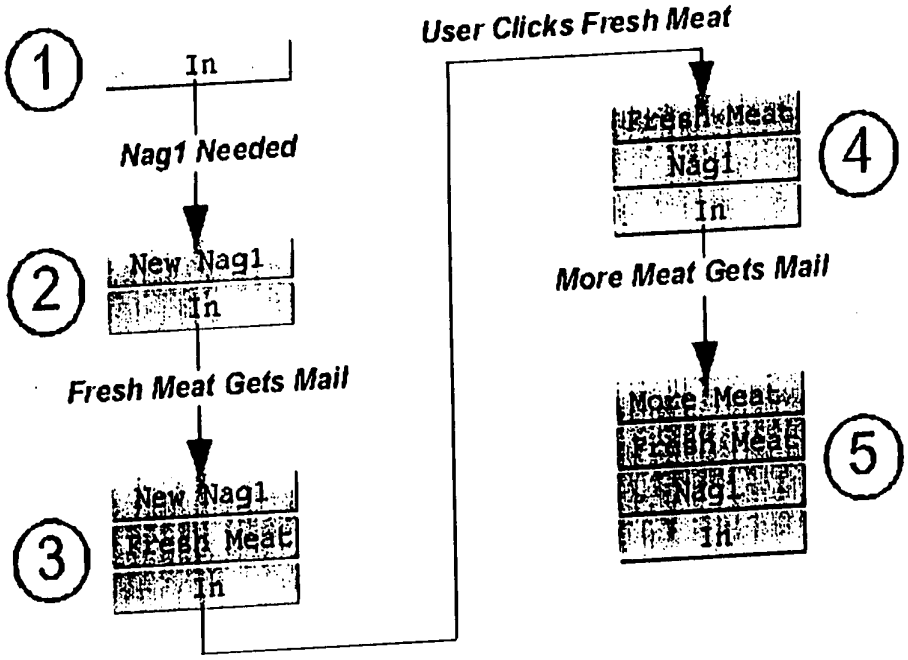

Fig. 10

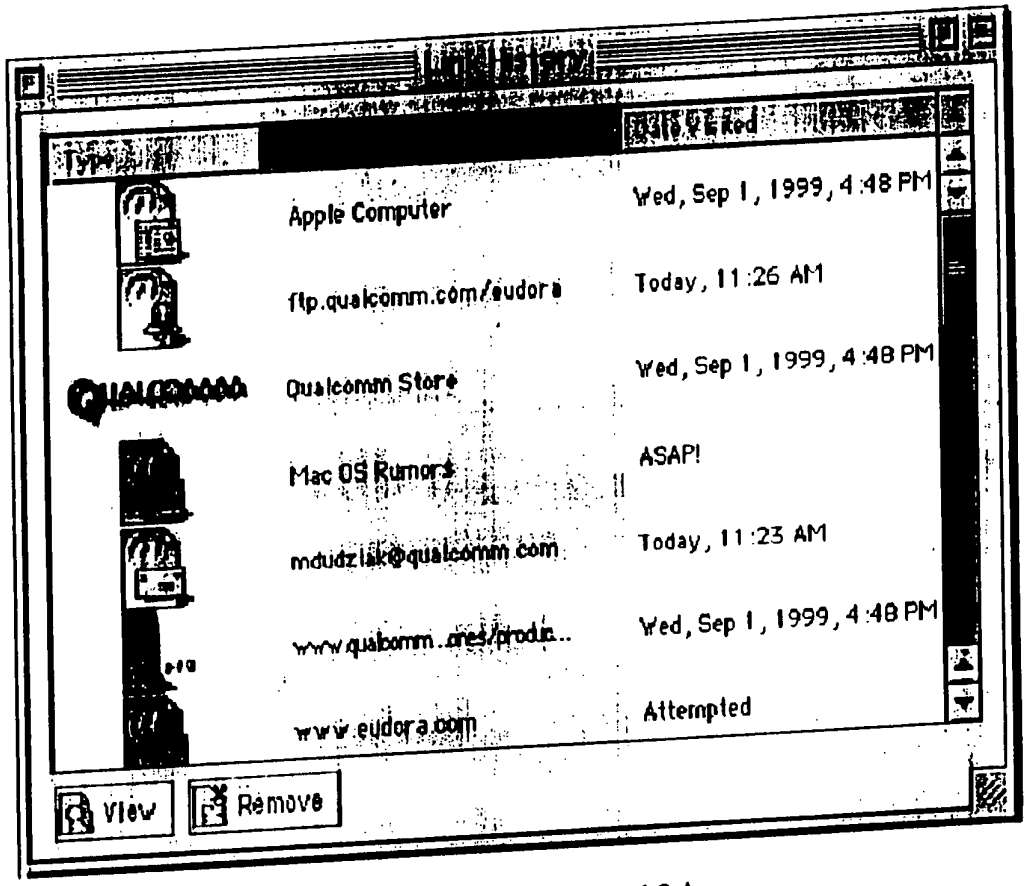
Apple Computer
Wed, Sep 1, 1999, 4:48 PM
ftp.qualcomm.com/eudora
Today, 11:26 AM
Qualcomm Store
Wed, Sep 1, 1999, 4:48 PM
Mac OS Rumors
ASAP!
mdudziak@qualcomm.com
Today, 11:23 AM
www.qualcomm...
Wed, Sep 1, 1999, 4:48 PM
www.eudora.com
Attempted
View
Remove

Fig. 12A

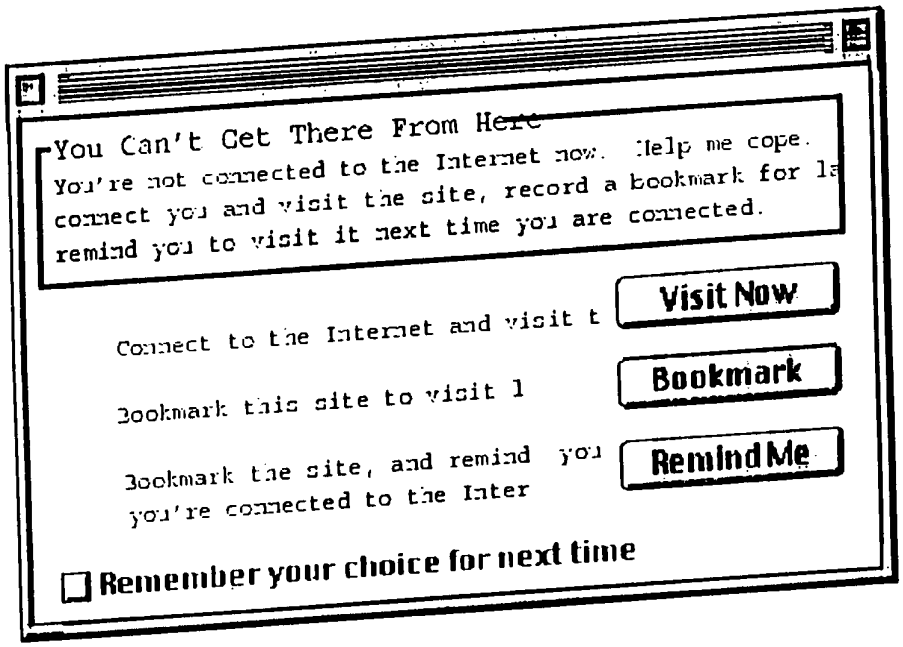

Fig. 12B

Assumptions	
Average Connect. Speed, Kbps	28.8
Average Ad Size, Kbytes	9.3
Number of Users	8,000,000
Number of Hours Running Eudora	2
Number Mailchecks Per User Per Hour	2
Playlist Entry Size, Bytes	500

Fig. 13A

Implications							
# of New Ads Per User Per Day	# Seconds Downloading Ads	# Seconds Added Per Check	8M Users Ad Bandwidth, Mbps	Ad Mbps / 100,000 users	Avg Sim. Connections, 1000's	8M Users Playlist Bandwidth, Mbps	Playlist Mbps / 100,000 users
10	26	[illegible]	[illegible]	0.8	2.4	[illegible]	[illegible]
15	39	10	101	1.3	3.6	5	0.1
20	52	13	135	1.7	4.8	7	0.1
25	65	[illegible]	[illegible]	[illegible]	6.0	[illegible]	[illegible]
30	78	19	202	2.5	7.2	11	0.1
35	90	23	235	2.9	8.4	12	0.2

Fig. 13B

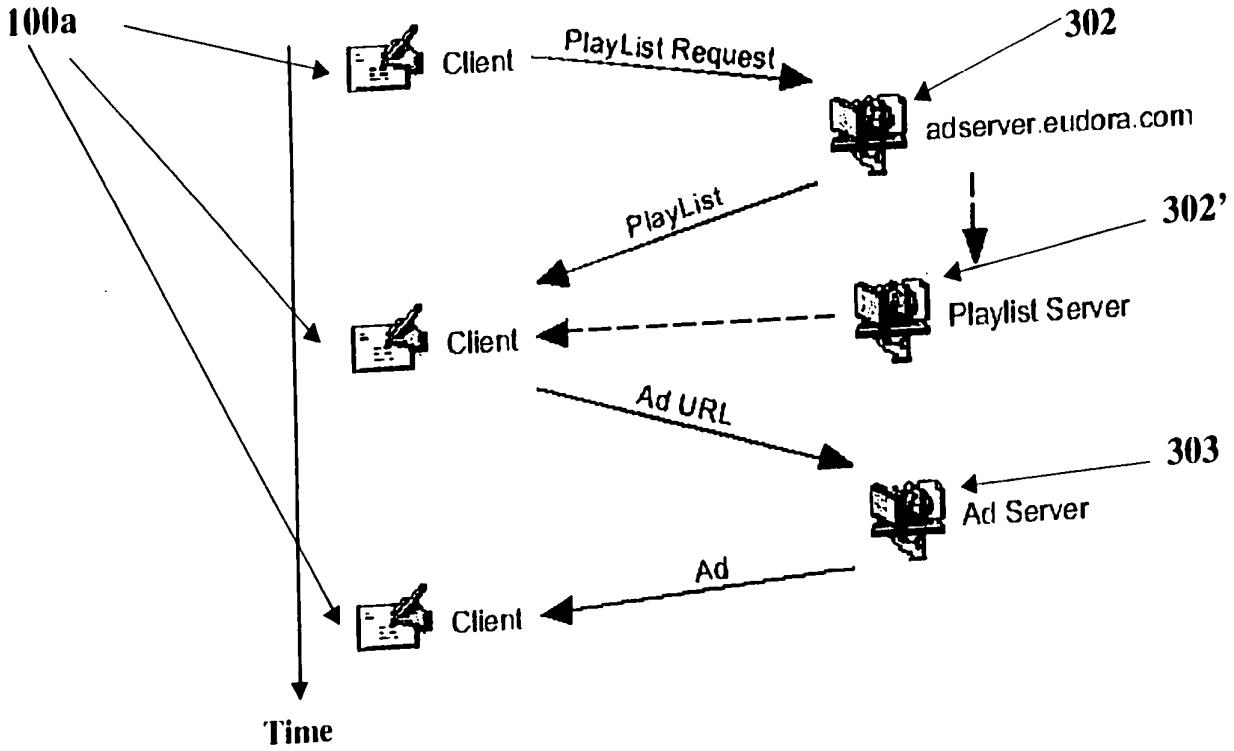

Fig. 14

```
////////////////////////////////////
// Main ad scheduler
ScheduleMain
{
// Has a new day dawned?
Do CheckForNewDay
// Are we are within the current ad's showFor?
if ( ad.thisShowTime < ad.showFor )
{
// there is nothing to be done
return
}
// At this point, we know that we need a new ad
// Perform housekeeping tasks on the old one
Do AdEndBookkeeping
// Pop out of a block if all ads on par
if ( block isn't all playlists )
{
find ad with minimum ad.numberShown
if ( ad.numberShown >= blockGoal )
set block to all playlists
}
// If we are over our quota of regular ads for the day,
// look for a runout
if ( adFaceTimeToday > faceTimeQuota )
{
Do ShowARunout
}
else
{
Do ShowARegularAd
}
}
// end ad schedule main
```

Fig. 15A

```
////////////////////////////////////
// We must perform certain tasks when the calendar day
changes.
CheckForNewDay
{if ( the calendar day has changed )
{
// Perform housekeeping tasks on the ad currently showing
Do StopShowingCurrentAd
// Runout ads are charged for a full showFor if they've been
shown
// at all on a given day. Charge any runout ads if they've
been
// shown at all.
for runout ads
{
if ( ad.thisShowTime > 0 )
{
ad.totalTimeShown += ad.showFor
ad.thisShowTime = 0
}
}
// Now, reset the counters for all ads to reflect the fact
that
// a new day has dawned.
for all ads
{
ad.numberShownToday = 0
}
// Record yesterday's facetime
// Might not literally be yesterday, be sure to use
// whatever day the app was last run on
set old current day's facetime to totalFaceTimeToday
// and reset our global regular ad facetime counter
adFaceTimeToday = 0
totalFaceTimeToday = 0
// if we were in a block, back out
set block to all playlists
}
}
// end CheckForNewDay
```

Fig. 15B

```
////////////////////////////////////
// This function shows a runout ad, and if it
// can't find one, goes to a rerun
ShowARunout
{
for runout ads
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this runout today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// are we done showing this runout for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next runout ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next runout ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this runout
// we are now in runout state
Do ShowAnAd
return
}
// if we haven't found a runout ad, we will go to "rerun"
state
Do ShowARerun
}
// end ShowARunout
```

Fig. 15C

```
////////////////////////////////////
// Rerun state. Look for a regular ad to rerun
ShowARerun
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// is this ad recent enough to rerun?
if ( ad.lastShownDate is older than returnInterval )
try next ad
// this one is too old to rerun
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, at this point we can show this ad, but because
// we're in rerun, we don't keep the books
Do ShowAnAd
return
}
// if we get here, we have no ads to show. Punt.
return
}
// end ShowARerun
```

Fig. 15D

```
////////////////////////////////////
// Show a regular ad
ShowARegularAd
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this ad today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we done showing this ad for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this ad
// we are now in regular state
Do ShowAnAd
return
}
// If we get here, we have failed to find a regular
// ad. Go to runout
Do ShowARunout
}
// end ShowARegularAd
```

Fig. 15E

```
////////////////////////////////////
// Perform necessary housekeeping when we're taking
// down an ad
AdEndBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// Account for at most ad.showFor seconds, provided
// we've shown the ad for at least ad.showFor seconds
// Note that this means we don't charge for time beyond
// ad.showFor seconds, which is important
if ( ad.thisShowTime >= ad.showFor )
{
ad.numberShownToday += ad.showFor
ad.shownFor++
// we do NOT reset thisShowTime here, we do it in
// AdStartBookkeeping. It actually doesn't matter where
// we do it, provided we are careful NOT to do it for
// runout ads.
}
}
// end AdEndBookkeeping
```

Fig. 15F

```
////////////////////////////////////
// Show an ad, including bookkeeping and block handling
ShowAnAd
{
// If the ad is in a block, notice that
if ( it's in a "block" playlist )
{
if ( not currently in a block )
{
find ad in block with minimum numberShown
make that our ad
set blockGoal to minimum numberShown+1
}
set current block to this playlist
}
// now do bookkeeping
Do AdStartBookkeeping
// and actually show it
Do DisplayThatAd
}
```

Fig. 15G

```
////////////////////////////////////
// Perform housekeeping when we put up an ad
AdStartBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// For regular ads
if ( it's a regular ad )
{
ad.thisShowTime = 0
ad.lastShownDate = now
}
}
// end AdStartBookkeeping
```

Fig. 15H

Persistent Ads		
PlayList Request	faceTime	Used to determine how much advertising to send to client
	faceTimeLeft	Not used
PlayList Response ClientInfo	reqInterval	Relatively large: one or more days
	flush	Used. Single playlist completely specifies list of ads client should have
PlayList Response Scheduling Parameters	showForMax	Not used

Fig. 16A

Short-Lived Ads		
PlayList Request	faceTime	Not used
	faceTimeLeft	Used to determine how many ads client should receive
PlayList Response ClientInfo	reqInterval	Not used. Instead, client requests new playlist whenever ads "run low".
	flush	Not used
PlayList Response Scheduling Parameters	showForMax	Used to determine how long an ad runs

Fig. 16B

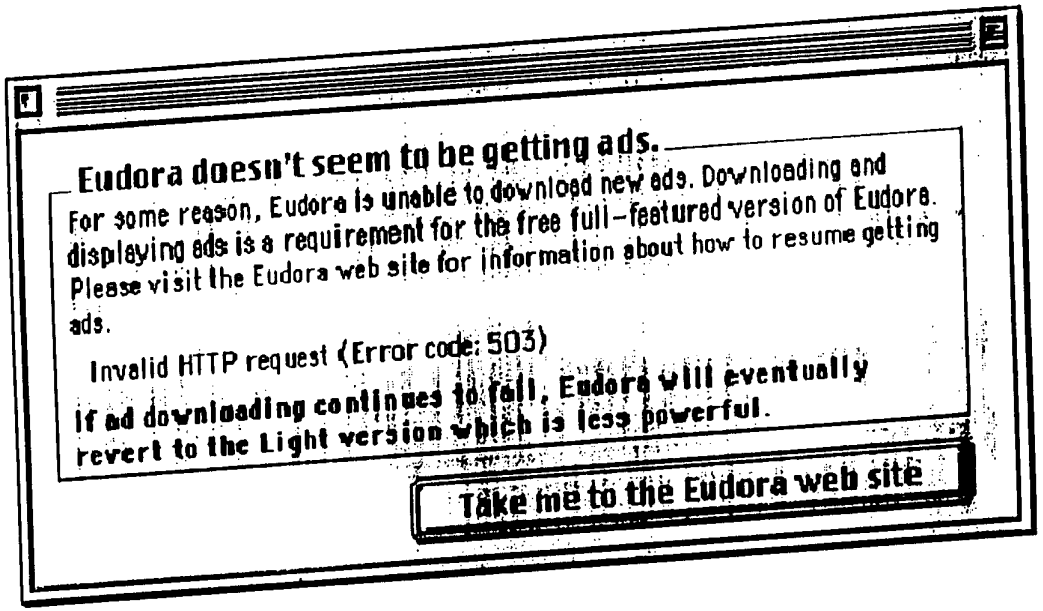

Fig. 17A

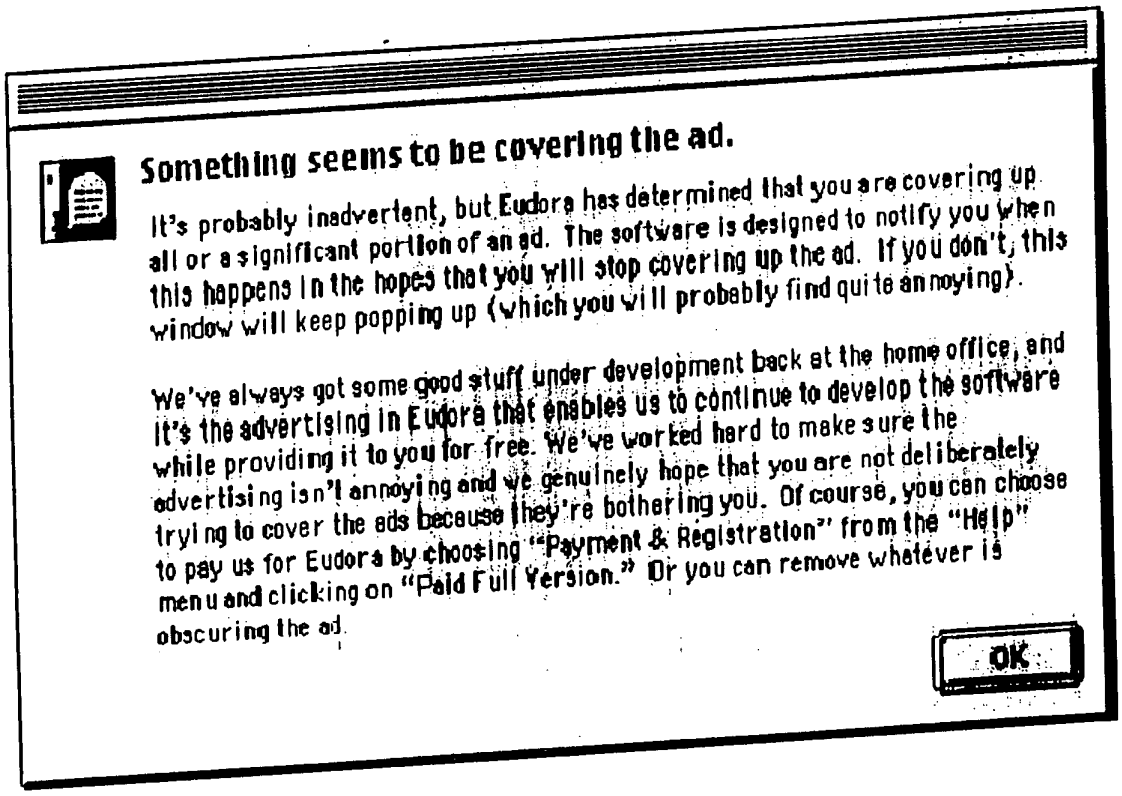

Fig. 17B

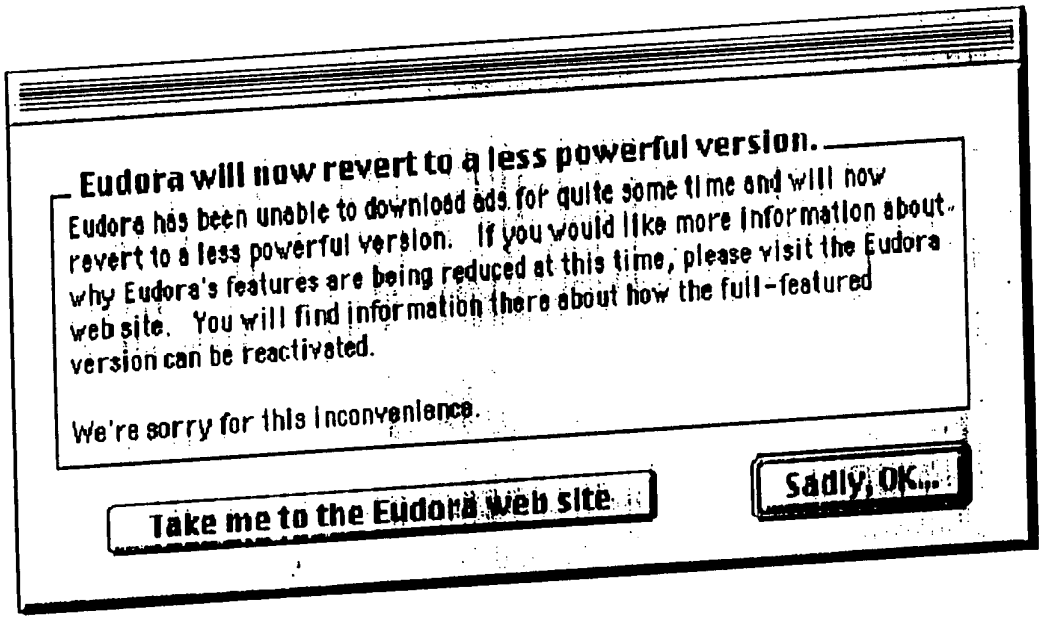

Fig. 17C

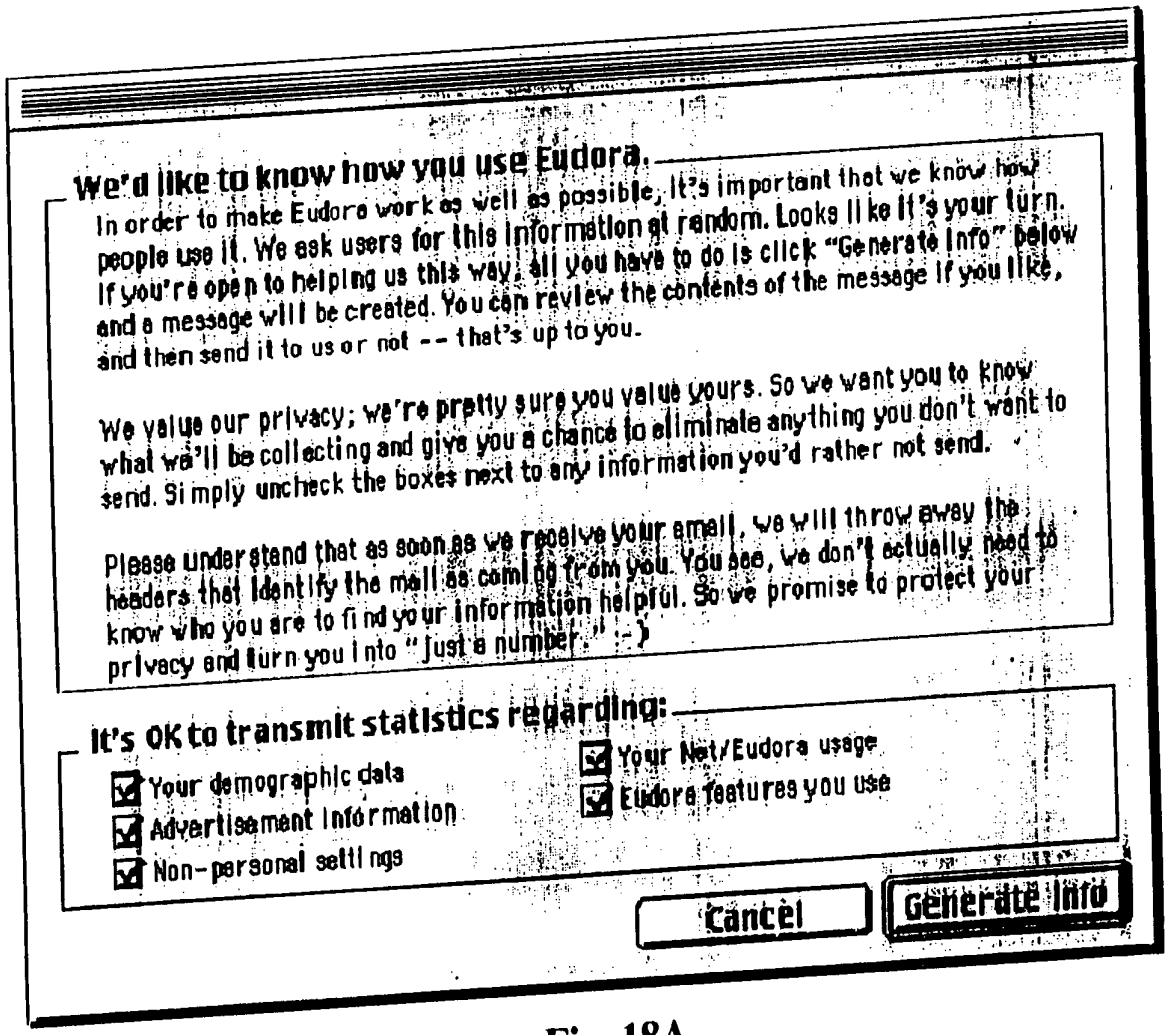
We'd like to know how you use Eudora.
In order to make Eudora work as well as possible, it's important that we know how people use it. We ask users for this information at random. Looks like it's your turn. If you're open to helping us this way, all you have to do is click "Generate Info" below and a message will be created. You can review the contents of the message if you like, and then send it to us or not -- that's up to you.
We value our privacy; we're pretty sure you value yours. So we want you to know what we'll be collecting and give you a chance to eliminate anything you don't want to send. Simply uncheck the boxes next to any information you'd rather not send.
Please understand that as soon as we receive your email, we will throw away the headers that identify the mail as coming from you. You see, we don't actually need to know who you are to find your information helpful. So we promise to protect your privacy and turn you into "just a number." :-)
It's OK to transmit statistics regarding:
Your demographic data
Advertisement information
Non-personal settings
Your Net/Eudora usage
Eudora features you use
Cancel
Generate Info

Fig. 18A

Page	Applicable Query Parts															
	action	platform	product	version	distributorID	mode	realname	email	regfirst	reglast	regcode	oldReg	regLevel	profile	url	adid	topic
Payment	pay	X	X	X	X	X	X	X	X	X	X	X					
Freeware Registratio	register-free	X	X	X	X	X	X	X	X	X	X						
Adware Registration	register-ad	X	X	X	X	X	X	X	X	X	X						
Box Registrations	register-box	X	X	X	X	X	X	X	X	X	X	X					
Lost Code	lostcode	X	X	X	X	X							X				
Update	update	X	X	X	X	X							X				
Pro Update	proupdate	X	X	X	X	X											
Archived	archived	X	X	X	X	X								X			
Profile	profile	X	X	X	X	X	X	X									
Introduction	intro	X	X	X	X	X	X	X	X	X	X	X					
Support	n/a	X	X	X	X												no-qt
QuickTime Missing	support	X	X	X	X												ad-fail
Ad Failure	support	X	X	X	X												tutor
Tutorial	support	X	X	X	X												faq
FAQ	support	X	X	X	X												light
Light Users	support	X	X	X	X												search
Search Support	support	X	X	X	X												usenet
Newsgroups	support	X	X	X	X												

Fig. 19

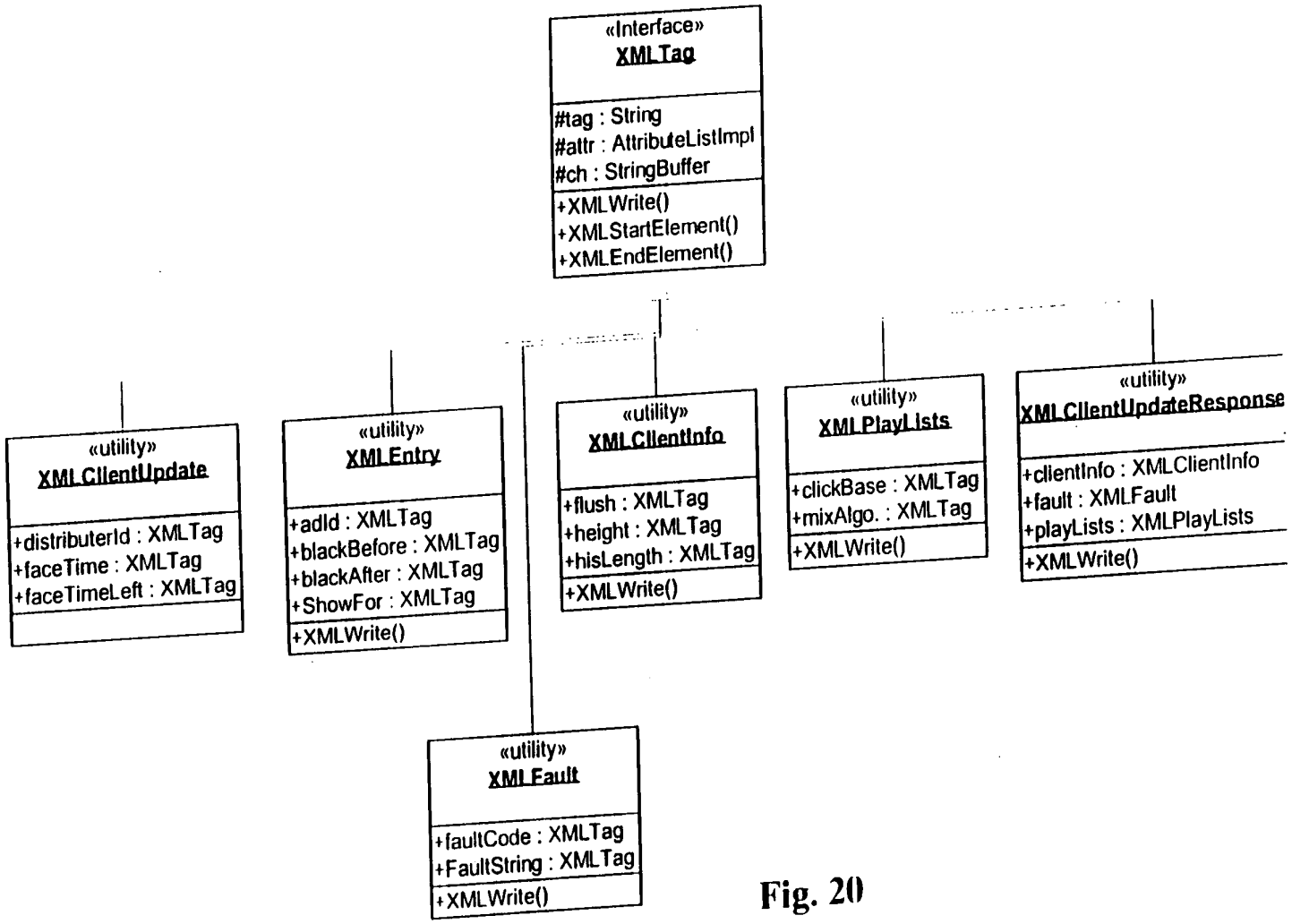

Fig. 20

- The list of available ads advantageously can be built from the following query:

ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = "I" AND AdStatus = "A" AND ImpressionsServed < Impressions ORDERD BY ImpressionsServed ASC);

run out ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = "R" AND AdStatus = "A" AND ImpressionsServed < Impressions ORDERD BY ImpressionsServed ASC);

- The time required to deliver the ads advantageously can be calculated in the following manner.

face time left for today [seconds] = faceTime[today] – faceTimeUsedToday

(Comment: Face time left for today is the number of secondes the servlet can use to deliver special ads today.)

predict face time [seconds] = SUM(faceTime[tomorrow] , faceTime[tomorrow + 1] , ... faceTime[tomorrow + reqInterval])

(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)

goal show time left [seconds] = predict face time – faceTimeLeft

(Comment: Goal show time left is the number of seconds that the software provider needs to fill with ads.)

Fig. 21A

```
Targeting
    while (face time left for today ) {
        if ad is not in the history {
            select ad [according to target = today]
            face time left for today -= ad.showFor
        }
        next ad
}

while (Goal show time left ) {
    if ad is not in the history {
        select ad [according to target]
        goal show time left -= ad.showFor
    }
    next ad
}

Default values:
    reqInterval = 1 day.
    facetime = 30 minutes
    faceTimeQuota is ?
    histLength = 31 days
```

Fig. 21B

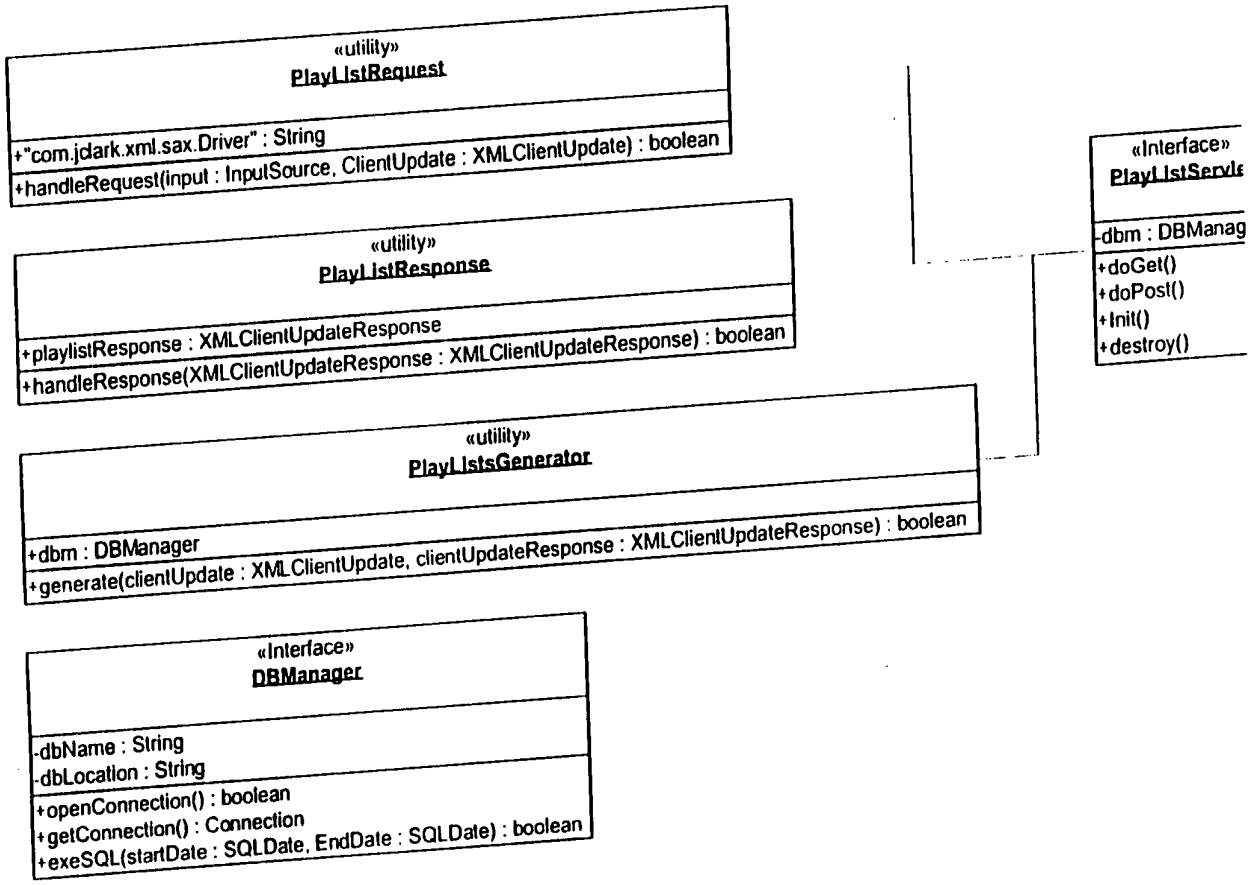
«utility»
PlayListRequest
+"com.jdark.xml.sax.Driver" : String
+handleRequest(input : InputSource, ClientUpdate : XMLClientUpdate) : boolean
«utility»
PlayListResponse
+playlistResponse : XMLClientUpdateResponse
+handleResponse(XMLClientUpdateResponse : XMLClientUpdateResponse) : boolean
«utility»
PlayListsGenerator
+dbm : DBManager
+generate(clientUpdate : XMLClientUpdate, clientUpdateResponse : XMLClientUpdateResponse) : boolean
«Interface»
DBManager
-dbName : String
-dbLocation : String
+openConnection() : boolean
+getConnection() : Connection
+exeSQL(startDate : SQLDate, EndDate : SQLDate) : boolean
«Interface»
-dbm : DBManag
+doGet()
+doPost()
+Init()
+destroy()

Fig. 22

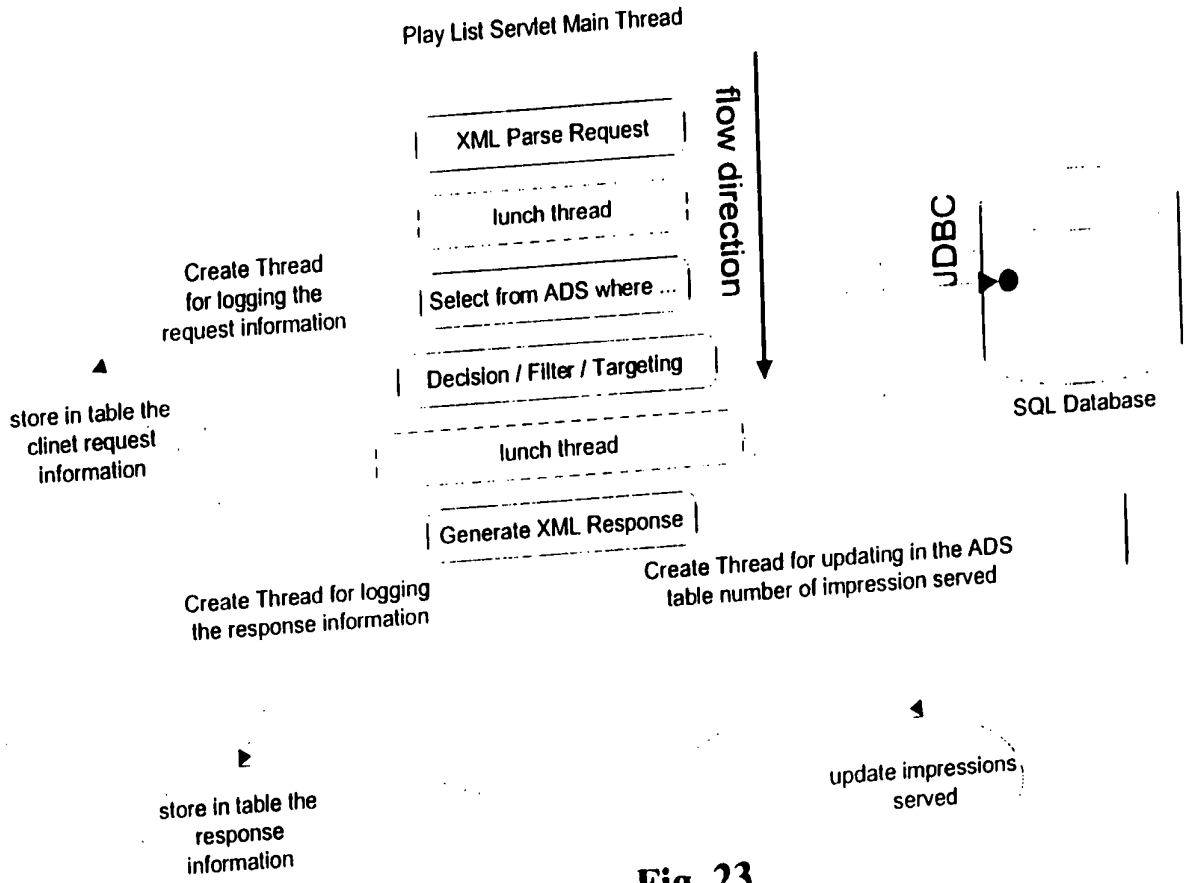

Fig. 23